ชจร โรจนเมธินทร์

SOFT MAIL

การวาดรูปกับรีเคอร์ชั่น

หลักการของโปรแกรมเรียกตนเองในลักษณะของภาพกราฟิก

การที่เราจะบอกว่าสิ่งใดมีคุณสมบัติรีเคอร์ชั่นได้นั้น สิ่งนั้นจะต้องมีส่วนประกอบส่วนหนึ่งเป็นตัวของมันเอง ตัวอย่างอันหนึ่งของรีเคอร์ชั่นที่เห็นได้ง่ายก็คือ ถ้าคุณนั่งอยู่ระหว่างกระจกเงาสองบานที่วางหันหน้าเข้าหากัน คุณจะเห็นภาพตัวเองสะท้อนกลับไปมาในกระจกเงาจนนับจำนวนได้ไม่ถ้วน รีเคอร์ชั่นส่วนมากจะใช้ในทางคณิตศาสตร์เป็นส่วนใหญ่เช่น ค่าของแฟกทอเรียลที่ N จะเป็นได้สองกรณีคือ

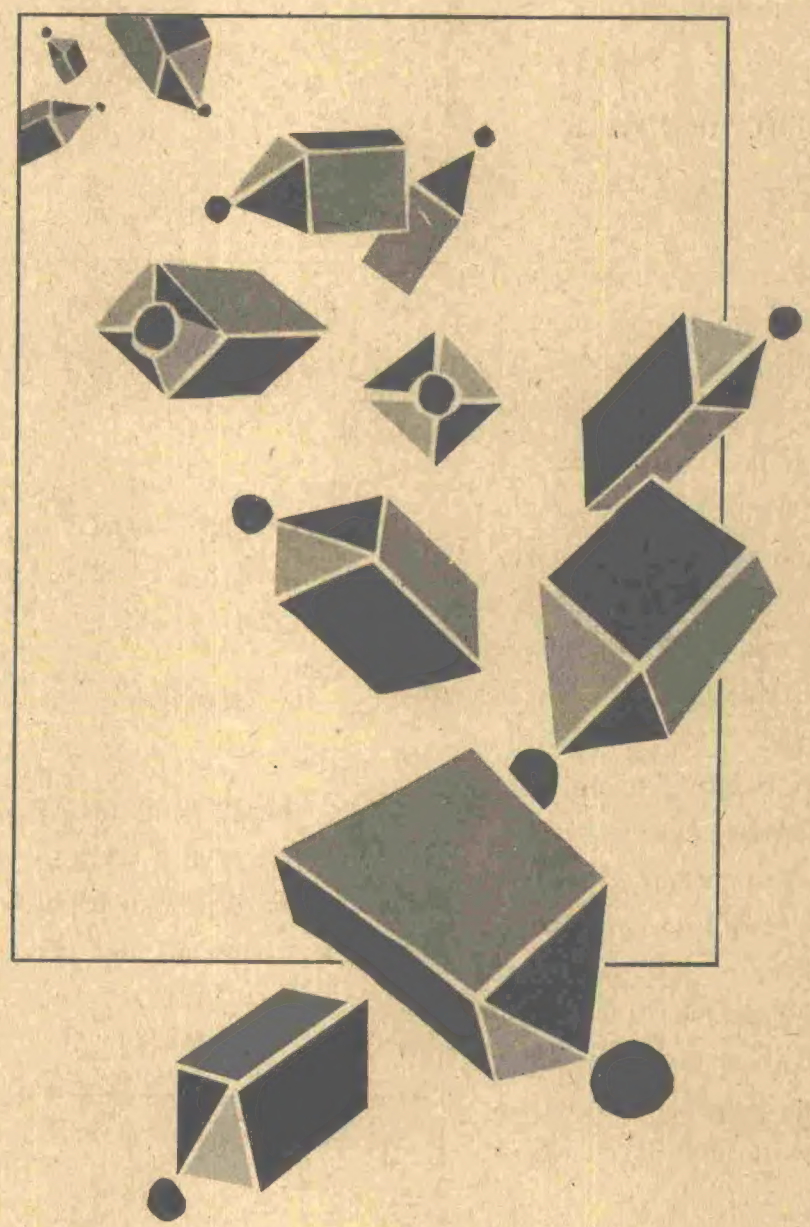
1. 0! = 1
2. N! = N(N−1)!; N > 0

จะเห็นว่าการจะหาค่าแฟกทอเรียลที่ N ได้นั้นจะต้องหาค่าของแฟกทอเรียลที่ (N−1) ได้ก่อน ในที่นี้ผมต้องขอยกตัวอย่างสำหรับผู้อ่านที่ไม่คุ้นเคยกับค่าแฟกทอเรียล สมมติว่าเราต้องการหาค่าแฟกทอเรียลของ 3 จะทำได้ดังนี้

3! = 3*(2!) = 3*2*(1!)
= 3*2*1*(0!) = 3*2*1*1 = 6

ดังนั้นค่าแฟกทอเรียลของ 3 จะเท่ากับ 6 นั่นก็คือการกำหนดค่าของแฟกทอเรียลใช้คุณสมบัติรีเคอร์ชั่นช่วย สำหรับในทางคณิตศาสตร์นั้นจะเขียนแทนความหมายของรีเคอร์ชั่นดังนี้

$P \equiv \wp[S_i, P]$



ถ้าเปรียบเทียบกับการหาค่าแฟกทอเรียลก็จะอธิบายได้ว่า ค่าของแฟกทอเรียล (P) นั้นจะประกอบด้วยสองส่วนคือ 0! (S_i) และค่าของแฟกทอเรียลของค่าที่มากกว่าศูนย์ (P)

การนำรีเคอร์ชั่นมาใช้ในการเขียนโปรแกรมนั้นเราจำเป็นต้องสร้างเป็นโปรแกรมย่อยขึ้นมาโดยมีการเรียกใช้โปรแกรมย่อยในส่วนของโปรแกรมย่อยนั้นเองเช่นจากรูปที่ 1 เป็นตัวอย่างโปร-

แกรมย่อยที่เขียนด้วยภาษาซีจะเห็นว่ามีการเรียกตัวเองมาทำงาน หลังจากคำสั่ง printf ซึ่งจะเรียกรีเคอร์ชั่นลักษณะนี้ว่า Direct recursion แต่ถ้าเป็นการเรียกตัวเองผ่านการเรียกโปรแกรมย่อยอื่นดังเช่นโปรแกรมย่อยรูปที่ 2 จะเรียกว่า Indirect recursion

สิ่งที่ควรทราบในการใช้รีเคอร์ชั่น

ปกติการสร้างโปรแกรมย่อยมาใช้งานนั้นมักจะต้องมีการกำหนดตัวแปรที่จะใช้ภายในโปรแกรมย่อย (Local variable) ดังนั้นเมื่อมีการเรียกใช้โปรแกรมย่อยที่มีการเรียกตัวเองแต่ละครั้งค่าของตัวแปรภายในดังกล่าวจะถูกเก็บไว้ในสแต็กเสมอแม้ว่าตัวแปรภายในของโปรแกรมย่อยเหล่านั้นจะมีชื่อเหมือนกัน และค่าที่เก็บไว้นี้จะนำกลับคืนมาเมื่อการทำงานของโปรแกรมย่อยครั้งสุดท้ายจบลง

สิ่งที่สำคัญมากอีกอย่างก็คือเราจำเป็นต้องสร้างจุดสิ้นสุดของการทำงานของโปรแกรมย่อยที่มีการเรียกตัวเองเสมอ มิฉะนั้นโปรแกรมย่อยจะเรียกตัวเองไปเรื่อย ๆ ไม่สิ้นสุดจนกว่าจะเกิดสแต็กโอเวอร์โฟลว์

การสร้างจุดสิ้นสุดนี้ส่วนใหญ่จะกำหนดเป็นเงื่อนไขเพื่อให้โปรแกรมย่อยมีการตรวจสอบก่อนที่จะเรียกตัวเองดังนี้

$P \equiv$ if B then $\wp[S_i, P]$ หรือ

$P \equiv \wp[S_i,$ if B then $P]$

สำหรับเงื่อนไข B ที่มักใช้ก็คือการส่งพารามิเตอร์ให้โปรแกรมย่อยโดยลดค่าลงแต่ละครั้งที่มีการเรียกโปรแกรมย่อย และตัวโปรแกรมย่อยก็จะตรวจสอบค่าพารามิเตอร์ก่อนที่จะเรียกตัวเอง เช่น

$P(n) \equiv$ if $n > 0$ then $\wp[S_i, P(n-1)]$ หรือ

$P(n) \equiv \wp[S_i,$ if $n > 0$ then $P(n-1)]$

```
void write_str()
{ int i = 2;
  printf("Directly recursive function");
  write_str();
}
```

รูปที่ 1 โปรแกรมย่อยแบบรีเคอร์ชั่นที่เรียกโดยตรง

```
void write_msg()
{ printf("Indirectly recursive function");
  write_scr();
}
void write_scr()
{ write_msg();
}
```

รูปที่ 2 โปรแกรมย่อยแบบรีเคอร์ชั่นที่เรียกโดยอ้อม

Prolog กับ Recursion

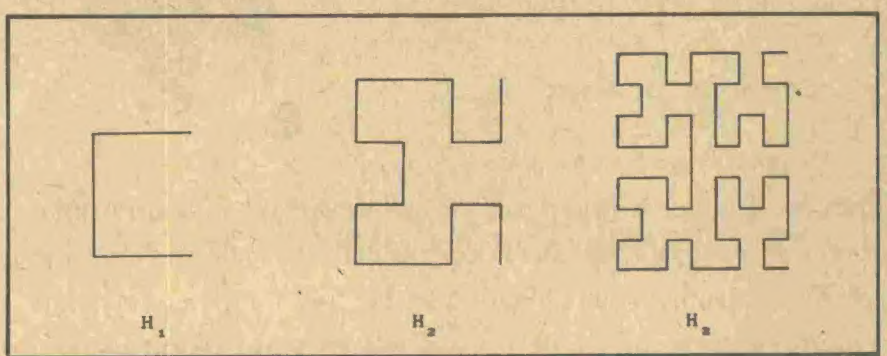
ภาษาคอมพิวเตอร์ทั่วไป เช่น Pascal, C, BASIC มักมีคำสั่งที่ใช้ทำงานวนรอบคือ FOR, WHILE, REPEAT แต่ในภาษา Prolog จะไม่มีคำสั่งที่ทำงานนี้โดยตรงเลย แต่อย่างไรก็ตามเราก็สามารถทำงานวนรอบในภาษา Prolog ได้ 2 วิธีคือ

1. ใช้การ Backtracking ในการค้นหา Fact ของ Prolog ช่วย
2. ใช้ Tail recursion

ความแตกต่างของการทำงานวนรอบสองวิธีนี้พอแยกได้คร่าว ๆ ว่าวิธีที่ 1 นั้นจะใช้เนื้อที่ของหน่วยความจำน้อย แต่ไม่สามารถส่งผ่านค่าของพารามิเตอร์ไปใช้ในการทำงานวนรอบแต่ละครั้ง ในขณะที่วิธีที่ 2 จะทำได้แต่ก็จะใช้เนื้อที่บนหน่วยความจำมากกว่า

การวาด Hilbert curves

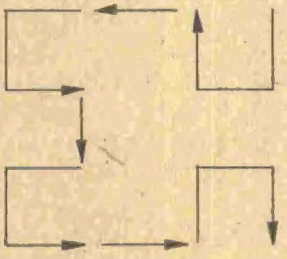
จากรูปที่ 3 แสดง Hilbert curves ลำดับที่ 1 ถึง 3 ถ้าคุณลองสังเกตดู



รูปที่ 3 Hilbert curves ลำดับที่ 1 ถึง 3

H_3 จะเห็นว่าประกอบด้วย H_2 จำนวน 4 รูป ต่อกันด้วยเส้นตรงโดยแต่ละรูปจะหมุนในทิศต่าง ๆ กัน, H_2 ก็จะประกอบด้วย H_1 จำนวน 4 รูป, H_1 คือเส้นตรงที่ต่อกัน 3 เส้น ดังนั้น H_i จะประกอบด้วย H_{i-1} จำนวน 4 รูป

คราวนี้มาพิจารณาขั้นตอนการวาดรูป ผมขอยกตัวอย่าง H_2 ซึ่งเราสามารถแยกรูป H_2 ออกเป็นส่วนย่อยได้ดังนี้



ถ้ากำหนดให้พรีดิเคต a, b, c, d ทำหน้าที่วาดส่วนย่อยของ H_2 ดังนี้

1. พรีดิเคต a วาดส่วน

2. พรีดิเคต b วาดส่วน

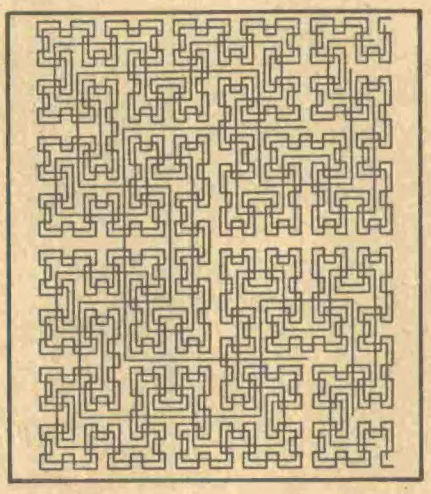
3. พรีดิเคต c วาดส่วน

4. พรีดิเคต d วาดส่วน

ดังนั้นขั้นตอนการวาด H_2 โดยการเรียกพรีดิเคตก็คือ

d — a ! a — b

โดยลูกศรแต่ละอันแทนการลากเส้นตรงเชื่อมรูปที่พรีดิเคตแต่ละอันวาด คราวนี้จากขั้นตอนการวาด H_2 นี้ ถ้าพรีดิเคตแต่ละอันไม่ทำงานคงมีแต่การลากเส้นเชื่อม รูปที่ได้ก็คือ H_1 ซึ่งพรีดิเคต a ทำหน้าที่วาดส่วนประกอบย่อยนี้ได้

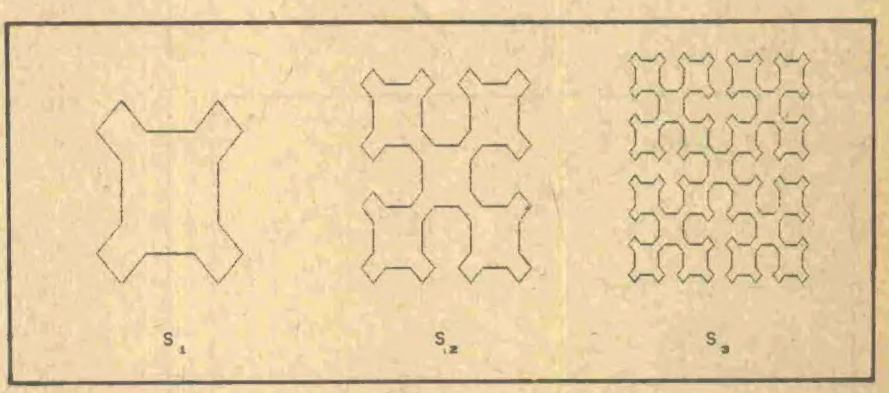


รูปที่ 4 Hilbert curves ที่วาดจากโปรแกรมรูปที่ 7

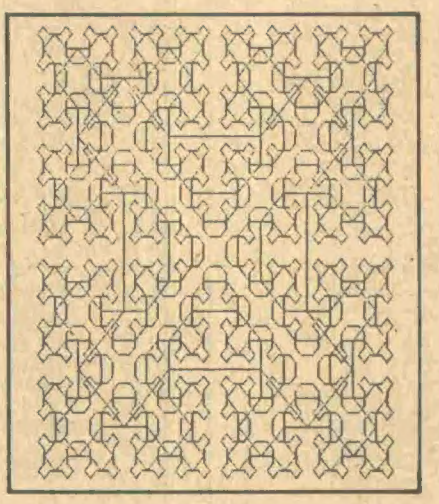
สรุปขั้นตอนการทำงานของพรีดิเคตแต่ละอันมีดังนี้

1. พรีดิเคต a : d — a ! a — b
2. พรีดิเคต b : c ! b — b ! a
3. พรีดิเคต c : b — c ! c — d
4. พรีดิเคต d : a ! d — d ! c

จากโปรแกรมในรูปที่ 7 ที่ทำหน้าที่วาด Hilbert curves นั้น ค่า hO จะแทนความกว้างของสี่เหลี่ยมจัตุรัสซึ่งเป็นพื้นที่ที่ใช้วาดรูป และ $hO = 2^k$ โดย k มีค่ามากกว่าหรือเท่ากับ n ซึ่งกำหนด n = 4 โปรแกรมก็จะวาด $H_1 ... H_{n+1}$ ซ้อนกันตามรูปที่ 4



รูปที่ 5 Sierpinski curves ลำดับที่ 1 ถึง 3



รูปที่ 6 Sierpinski curves ที่วาดจากโปรแกรมรูปที่ 8

สำหรับ Facts ที่เก็บในดาต้าเบสได้แก่

1. cur_xy จะช่วยเก็บค่าพอยน์เตอร์ปัจจุบันของจอภาพในการวาดรูป
2. right_xy ใช้เก็บจุดตั้งต้นที่จะใช้เริ่มลากเส้นในการวาดรูปแต่ละครั้ง

สำหรับพรีดิเคต hilbert จะคอยควบคุมการวาดรูปแต่ละครั้งโดยกำหนดขนาดของรูป (HI) และจุดเริ่มต้น (XI, YI) จากนั้นก็เริ่มต้นวาดรูปด้วยพรีดิเคต a ซึ่งมีการเรียกตัวเองทั้งแบบโดยตรงและอ้อม

จากโปรแกรมในรูปที่ 8 จะทำหน้าที่วาด Sierpinski curves ที่แสดงในรูปที่ 6 โดยแยกออกเป็นส่วนย่อยตามรูปที่ 5 ส่วนการทำงานคล้ายกับโปรแกรมในรูปที่ 7

ก่อนจบ

บทความนี้ผมตั้งใจจะเสนอแนวทางการเขียนโปรแกรมแบบรีเคอร์ชั่น ซึ่งการเขียนโปรแกรมแบบนี้ช่วยทำให้การเขียนโปรแกรมเข้าใจง่ายและสั้นด้วย แต่ก็จำกัดกับงานที่มีโครงสร้างเป็นแบบรีเคอร์ชั่นเท่านั้น หวังว่าการเขียนโปรแกรมแบบรีเคอร์ชั่นนี้คงเป็นประโยชน์กับผู้อ่านในการเขียนโปรแกรมครั้งต่อไป

รูปที่ 7 โปรแกรม Hilbert curves

```
include "A:GRAPDECL.PRO"

constants
  n = 4
  h0 = 256
  bgi_path = "a:"

domains
  type = integer

database - mydata
  cur_xy(type,type)
  right_xy(type,type)

predicates
  hilbert(type,type)
  a(type,type)
  b(type,type)
  c(type,type)
  d(type,type)

goal
  initgraph(0,0,_,_,bgi_path),
  I = 0, H = h0,
  X = H div 2, Y = X,
  settextstyle(triplex_Font,horiz_Dir,4),
  outtextxy(350,80,"Hilbert Curves."),
  outtextxy(350,120,"n = 4, h0 = 256"),
  retractall(_,mydata),
  assert(cur_xy(X,Y)),
  assert(right_xy(X,Y)),
  hilbert(I,H),
  outtextxy(160,300,"Press any key to exit."),
  readchar(_),
  closegraph.

clauses
  hilbert(I,H) :-
        I <= n, !, I1 = I + 1,
        H1 = H div 2,
        right_xy(X,Y), X1 = X + (H1 div 2), Y1 = Y - (H1 div 2),
        retract(right_xy(X,Y)), assert(right_xy(X1,Y1)),
        retractall(cur_xy(_,_)), assert(cur_xy(X1,Y1)),
        moveto(X1,Y1),
        a(I1,H1), !,
        hilbert(I1,H1).
```

→

```
hilbert(_,_).

a(I,H) :-
      I > 0, !, I1 = I - 1,

      d(I1,H),
      cur_xy(X1,Y1), X = X1 - H,
      retract(cur_xy(X1,Y1)), assert(cur_xy(X,Y1)),
      lineto(X,Y1),

      a(I1,H),
      cur_xy(X2,Y2), Y = Y2 + H,
      retract(cur_xy(X2,Y2)), assert(cur_xy(X2,Y)),
      lineto(X2,Y),

      a(I1,H),
      cur_xy(X3,Y3), X4 = X3 + H,
      retract(cur_xy(X3,Y3)), !, assert(cur_xy(X4,Y3)),
      lineto(X4,Y3),

      b(I1,H).

a(_,_).

b(I,H) :-
      I > 0, !, I1 = I - 1,

      c(I1,H),
      cur_xy(X1,Y1), Y = Y1 - H,
      retract(cur_xy(X1,Y1)), assert(cur_xy(X1,Y)),
      lineto(X1,Y),

      b(I1,H),
      cur_xy(X2,Y2), X = X2 + H,
      retract(cur_xy(X2,Y2)), assert(cur_xy(X,Y2)),
      lineto(X,Y2),

      b(I1,H),
      cur_xy(X3,Y3), Y4 = Y3 + H,
      retract(cur_xy(X3,Y3)), !, assert(cur_xy(X3,Y4)),
      lineto(X3,Y4),

      a(I1,H).

b(_,_).

c(I,H) :-
      I > 0, !, I1 = I - 1,

      b(I1,H),
      cur_xy(X1,Y1), X = X1 + H,
      retract(cur_xy(X1,Y1)), assert(cur_xy(X,Y1)),
      lineto(X,Y1),

      c(I1,H),
      cur_xy(X2,Y2), Y = Y2 - H,
      retract(cur_xy(X2,Y2)), assert(cur_xy(X2,Y)),
      lineto(X2,Y),

      c(I1,H),
      cur_xy(X3,Y3), X4 = X3 - H,
      retract(cur_xy(X3,Y3)), !, assert(cur_xy(X4,Y3)),
      lineto(X4,Y3),

      d(I1,H).

c(_,_).

d(I,H) :-
      I > 0, !, I1 = I - 1,

      a(I1,H),
      cur_xy(X1,Y1), Y = Y1 + H,
      retract(cur_xy(X1,Y1)), assert(cur_xy(X1,Y)),
      lineto(X1,Y),

      d(I1,H),
      cur_xy(X2,Y2), X = X2 - H,
      retract(cur_xy(X2,Y2)), assert(cur_xy(X,Y2)),
      lineto(X,Y2),

      d(I1,H),
      cur_xy(X3,Y3), Y4 = Y3 - H,
      retract(cur_xy(X3,Y3)), !, assert(cur_xy(X3,Y4)),
      lineto(X3,Y4),

      c(I1,H).

d(_,_).
```

รูปที่ 8 โปรแกรม Sierpinski curves

```
include "A:GRAPDECL.PRO"

constants
  n = 3
  h0 = 256
  bgi_path = "a:"

domains
  type = integer

database - mydata
  cur_xy(type,type)
  right_xy(type,type)

predicates
  sierpinski(type,type)
  a(type,type)
  b(type,type)
  c(type,type)
  d(type,type)

goal
  initgraph(0,0,_,_,bgi_path),
  I = 0, H = h0 div 4,
  X = 2*H, Y = 3*H,
  settextstyle(triplex_Font,horiz_Dir,4),
  outtextxy(350,80,"Sierpinski Curves."),
  outtextxy(350,120,"n = 3, h0 = 256"),
  retractall(_,mydata),
  assert(cur_xy(X,Y)),
  assert(right_xy(X,Y)),
  sierpinski(I,H),
  outtextxy(160,300,"Press any key to exit."),
  readchar(_),
  closegraph.

clauses
  sierpinski(I,H) :-
        I <= n, !, I1 = I + 1,
        right_xy(X,Y), X1 = X - H, H1 = H div 2, Y1 = Y + H1,
        retract(right_xy(X,Y)), assert(right_xy(X1,Y1)),
        retractall(cur_xy(_,_)), assert(cur_xy(X1,Y1)),
        moveto(X1,Y1),

        a(I1,H1), !,
        cur_xy(X2,Y2), X3 = X2 + H1, Y3 = Y2 - H1,
        retract(cur_xy(X2,Y2)), assert(cur_xy(X3,Y3)),
        lineto(X3,Y3),

        b(I1,H1), !,
        cur_xy(X4,Y4), X5 = X4 - H1, Y5 = Y4 - H1,
        retract(cur_xy(X4,Y4)), assert(cur_xy(X5,Y5)),
        lineto(X5,Y5),

        c(I1,H1), !,
        cur_xy(X6,Y6), X7 = X6 - H1, Y7 = Y6 + H1,
        retract(cur_xy(X6,Y6)), assert(cur_xy(X7,Y7)),
        lineto(X7,Y7),

        d(I1,H1), !,
        cur_xy(X8,Y8), X9 = X8 + H1, Y9 = Y8 + H1,
        retract(cur_xy(X8,Y8)), !, assert(cur_xy(X9,Y9)),
        lineto(X9,Y9),

        sierpinski(I1,H1).

  sierpinski(_,_).

  a(I,H) :-
        I > 0, !, I1 = I - 1,

        a(I1,H),
        cur_xy(X1,Y1), X = X1 + H, Y = Y1 - H,
        retract(cur_xy(X1,Y1)), assert(cur_xy(X,Y)),
        lineto(X,Y),

        b(I1,H),
        cur_xy(X2,Y2), X3 = X2 + 2*H,
        retract(cur_xy(X2,Y2)), assert(cur_xy(X3,Y2)),
        lineto(X3,Y2),

        d(I1,H),
        cur_xy(X4,Y4), X5 = X4 + H, Y5 = Y4 + H,
        retract(cur_xy(X4,Y4)), !, assert(cur_xy(X5,Y5)),
        lineto(X5,Y5),

        a(I1,H).

  a(_,_).

  b(I,H) :-
        I > 0, !, I1 = I - 1,

        b(I1,H),
        cur_xy(X1,Y1), X = X1 - H, Y = Y1 - H,
        retract(cur_xy(X1,Y1)), assert(cur_xy(X,Y)),
        lineto(X,Y),

        c(I1,H),
        cur_xy(X2,Y2), Y3 = Y2 - 2*H,
        retract(cur_xy(X2,Y2)), assert(cur_xy(X2,Y3)),
        lineto(X2,Y3),

        a(I1,H),
        cur_xy(X4,Y4), X5 = X4 + H, Y5 = Y4 - H,
        retract(cur_xy(X4,Y4)), !, assert(cur_xy(X5,Y5)),
        lineto(X5,Y5),

        b(I1,H).

  b(_,_).

  c(I,H) :-
        I > 0, !, I1 = I - 1,

        c(I1,H),
        cur_xy(X1,Y1), X = X1 - H, Y = Y1 + H,
        retract(cur_xy(X1,Y1)), assert(cur_xy(X,Y)),
        lineto(X,Y),

        d(I1,H),
        cur_xy(X2,Y2), X3 = X2 - 2*H,
        retract(cur_xy(X2,Y2)), assert(cur_xy(X3,Y2)),
        lineto(X3,Y2),

        b(I1,H),
        cur_xy(X4,Y4), X5 = X4 - H, Y5 = Y4 - H,
        retract(cur_xy(X4,Y4)), !, assert(cur_xy(X5,Y5)),
        lineto(X5,Y5),

        c(I1,H).

  c(_,_).

  d(I,H) :-
        I > 0, !, I1 = I - 1,

        d(I1,H),
        cur_xy(X1,Y1), X = X1 + H, Y = Y1 + H,
        retract(cur_xy(X1,Y1)), assert(cur_xy(X,Y)),
        lineto(X,Y),

        a(I1,H),
        cur_xy(X2,Y2), Y3= Y2 + 2*H,
        retract(cur_xy(X2,Y2)), assert(cur_xy(X2,Y3)),
        lineto(X2,Y3),

        c(I1,H),
        cur_xy(X4,Y4), X5 = X4 - H, Y5 = Y4 + H,
        retract(cur_xy(X4,Y4)), !, assert(cur_xy(X5,Y5)),
        lineto(X5,Y5),

        d(I1,H).

  d(_,_).
```